

BI Lab Mini Project

Mobile Appstore games

Shrineeth Kotian - XIEIT181925
Manish Kumavat - XIEIT181926
Dixit Patel -XIEIT181936

May 09, 2021

List of Figures

1.1	Size of the appstore games dataset	4
2.1	Loading the dataset	7
2.2	Copying data into another variable	8
2.3	Missing Values	8
2.4	Average user rating	8
2.5	Average user rating (0 missing values)	9
2.6	User rating count	10
2.7	Price	11
2.8	Languages	11
2.9	Size	12
2.10	Information about dataset	13
2.11	Countplot graph 1	13
2.12	Countplot graph 2	14
2.13	Countplot graph 3	15
2.14	Countplot graph 4	16
2.15	Genre Attribute Information	16
2.16	Countplot graph 5	17
2.17	Lineplot graph 1	18
2.18	Lineplot graph 2	19
2.19	Lineplot graph 3	20
2.20	Lineplot graph 4	21
2.21	Lineplot graph 5	22
2.22	Scatterplot graph 1	23
2.23	Scatterplot graph 2	23
2.24	Scatterplot graph 3	24
2.25	Importing libraries	25
2.26	Displaying data	26
2.27	Dropping columns	26
2.28	Displaying columns	27
2.29	Displaying Description	27
2.30	Displaying Description after cleaning	28
2.31	Vectorizer method	28
2.32	Train And split data	29
2.33	Naïve Bayes classifier model	29
2.34	Naïve Bayes output	30
2.35	Tuning hyperparameters	31

2.36	Best Score (Naïve Bayes)	31
2.37	Decision Tree Classifier model	31
2.38	Decision Tree Output	32
2.39	Best Score (Decision Tree Classifier)	32
2.40	K-Nearest Neighbors Classifier model	33
2.41	K-Nearest Neighbors Classifier Output	33
2.42	Best Score (K-Nearest Neighbors Classifier)	34
2.43	Tuning of KNN Model	34
2.44	Graph of impact of the number of neighbors on the score of the model	35
2.45	Improved Accuracy of KNN	35
2.46	Code for Kmeans Clustering	36
2.47	Before applying kmeans	37
2.48	After applying kmeans	38
2.49	Importing libraries for DBSCAN	39
2.50	Plotting graph before applying clustering	39
2.51	Preparing Model	40
2.52	Visualizing results-1	41
2.53	Visualizing results-2	42
2.54	Plotting clusters and outliers	43
2.55	DBSCAN Output	43

Contents

1	Mobile Appstore Games:	4
1.1	Problem statement:	4
1.2	Overview of Dataset:	4
1.2.1	Size of the dataset:	4
1.2.2	Attributes of our dataset:	5
1.2.3	Attributes details:	5
1.3	Purpose of the dataset:	6
1.4	Steps in implementation of the project:	6
2	Implementation:	7
2.1	Loading the dataset:	7
2.2	Pre-Processing:	7
2.3	Data Exploration:	12
2.4	Classification Algorithms:	25
2.4.1	Naive Bayes Classifier:	29
2.4.2	Decision Tree Classifier:	31
2.4.3	K-Nearest Neighbors Classifier:	33
2.5	Clustering Algorithms:	35
2.5.1	Kmeans Clustering:	35
2.5.2	DBSCAN Clustering:	39
3	Conclusion	44
4	References	45

Mobile Appstore Games:

1.1 Problem statement:

The analysis of the appstore games dataset is hoped to indicate the overall success of a game, and then work out what factors make a successful game and try to predict where it is headed.

1.2 Overview of Dataset:

The mobile games industry is worth billions of dollars, with companies spending vast amounts of money on the development and marketing of these games to an equally large market. Using this data set, insights can be gained into a sub-market of this market, strategy games. This sub-market includes titles such as Clash of Clans, Plants vs Zombies and Pokemon GO.

This is the data of 17007 strategy games on the Apple App Store. It was collected on the 3rd of August 2019, using the iTunes API and the App Store sitemap.

You could use the number of ratings as a proxy indicator for the overall success of a game, and then work out what factors make a successful game. Or you could measure the state of the market over time and try predict where it is headed. This dataset has a lot of data about mobile apps, including the rating. What we don't know is when and why an app will get a good rating.

The objective of this task is to train a model that can predict the average user rating of an app.

Link to the dataset: <https://www.kaggle.com/tristan581/17k-apple-app-store-strategy-games>

1.2.1 Size of the dataset:

There are 17007 Rows and 16 Columns

```
[ ] data1.shape
(17007, 16)
```

Figure 1.1: Size of the appstore games dataset

1.2.2 Attributes of our dataset:

1. URL,
2. ID,
3. Name,
4. Icon URL,
5. Average User Rating,
6. User Rating count,
7. Price,
8. Developer,
9. Age Rating,
10. Languages,
11. Size,
12. Primary Genres,
13. Original Release Date,
14. Current Version Release Date ,
15. Description,
16. Genres.

1.2.3 Attributes details:

URL: The URL

ID: The assigned ID to a particular game

Name: The name of the game

Icon URL: 512px x 512px jpg

Average User Rating: Rounded to nearest .5, requires at least 5 ratings

User Rating Count: Number of ratings internationally, null means it is below 5

Price: Price in USD

Description: App description

Developer: App developer

Age Rating: Either 4+, 9+, 12+ or 17+

Languages: ISO2A language codes

Size: Size of the app in bytes

Primary Genre: Main genre

Genres: Genres of the app

Original Release Date': When it was released

Current Version Release Date: When it was last updated

1.3 Purpose of the dataset:

- The best Appstore games according to the User ratings
- To predict the maximum age ratings of each game
- To predict which language games do most users prefer purchasing
- To predict the average user ratings and whether it depends on the size

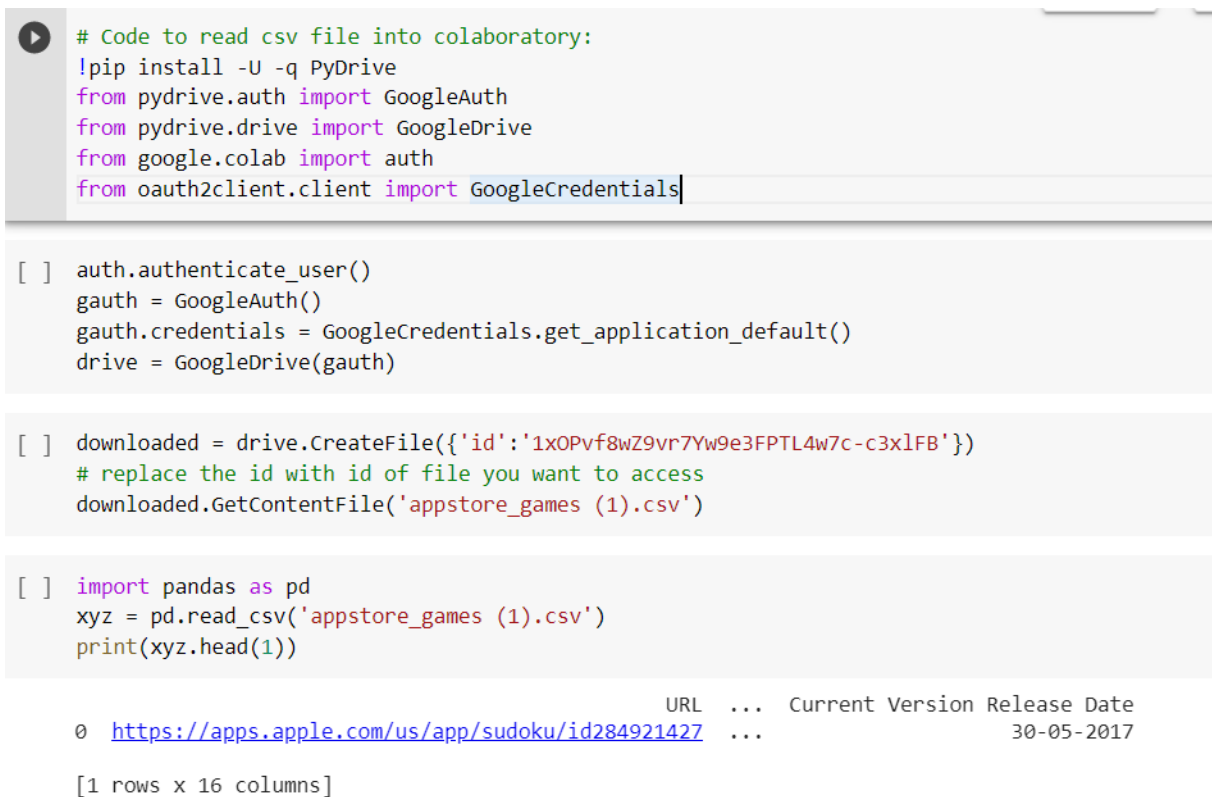
1.4 Steps in implementation of the project:

1. Perform Cleaning(Pre-Processing dataset)
2. Data Exploration (Statistical analysis of data and to find the relations between attributes)
3. Implementing Classification and Clustering Algorithms on our Mobile Appstore Games Dataset.
4. Calculating accuracy, confusion matrix of all the algorithms.
5. Comparing the performance of all the Algorithms from each category and selecting the best Algorithm from each category for prediction of your selected dataset.

Implementation:

2.1 Loading the dataset:

We have performed our project on the Google Collab platform, and the following figure will depict the loading of our dataset which is stored in the drive.



```
# Code to read csv file into colab:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

[ ] auth.authenticate_user()
    gauth = GoogleAuth()
    gauth.credentials = GoogleCredentials.get_application_default()
    drive = GoogleDrive(gauth)

[ ] downloaded = drive.CreateFile({'id':'1xOPvf8wZ9vr7Yw9e3FPTL4w7c-c3xlFB'})
    # replace the id with id of file you want to access
    downloaded.GetContentFile('appstore_games (1).csv')

[ ] import pandas as pd
    xyz = pd.read_csv('appstore_games (1).csv')
    print(xyz.head(1))

                                URL ... Current Version Release Date
0  https://apps.apple.com/us/app/sudoku/id284921427 ...                30-05-2017

[1 rows x 16 columns]
```

Figure 2.1: Loading the dataset

2.2 Pre-Processing:

In this we find the number of missing values in the dataset. And then comes the filling of the missing values using the methods such as mean, median, mode and other methods.

In the dataset there are 9447 and 9446 missing values in average user rating and user rating count respectively, 24 values in price 60 values in languages and only in size are missing values.


```
[ ] import pandas as pd
    data = pd.read_csv('appstore_games (1).csv')
    data1=data.copy()
```

Figure 2.2: Copying data into another variable

```
[ ] data1.isna().sum()

URL                                0
ID                                0
Name                              0
Icon URL                          0
Average User Rating              9447
User Rating Count                9446
Price                            24
Description                       0
Developer                        0
Age Rating                       0
Languages                        60
Size                              1
Primary Genre                     0
Genres                            0
Original Release Date             0
Current Version Release Date      0
dtype: int64
```

Figure 2.3: Missing Values

Starting with average user rating we decided to fill its missing values with mode of the field as the values were repeating, we get mode as 4.5 and that is inserted into dataset by using fillna function.

```
[ ] missing=data1[data1.isnull().any(axis=1)]
    data1['Average User Rating'].mode()

0    4.5
dtype: float64

[ ] data1['Average User Rating'].fillna(data1['Average User Rating'].mode()[0],inplace=True)
```

Figure 2.4: Average user rating

```
[ ] data1.isna().sum()

URL                                0
ID                                0
Name                              0
Icon URL                          0
Average User Rating               0
User Rating Count                 9446
Price                             24
Description                       0
Developer                         0
Age Rating                       0
Languages                         60
Size                              1
Primary Genre                     0
Genres                           0
Original Release Date             0
Current Version Release Date      0
dtype: int64
```

Figure 2.5: Average user rating (0 missing values)

The column Average user rating is free from missing values

For User Rating Count we calculate the mean value of the attribute which comes to be 3306, we add it to the dataset accordingly

```
[ ] data1['User Rating Count'].mean()
```

```
3306.5312789313584
```

```
[ ] data1['User Rating Count'].fillna(data1['User Rating Count'].mean(),inplace=True)  
data1.isna().sum()
```

```
URL          0  
ID           0  
Name         0  
Icon URL     0  
Average User Rating  0  
User Rating Count  0  
Price        24  
Description  0  
Developer    0  
Age Rating   0  
Languages    60  
Size         1  
Primary Genre  0  
Genres        0  
Original Release Date  0  
Current Version Release Date  0  
dtype: int64
```

Figure 2.6: User rating count

The column User rating count has 0 missing values

Similarly, calculating the mode of price column missing values are filled

```
[ ] data1['Price'].mode()

0    0.0
dtype: float64
```

```
[ ] data1['Price'].fillna(data1['Price'].mode()[0],inplace=True)
data1.isna().sum()
```

URL	0
ID	0
Name	0
Icon URL	0
Average User Rating	0
User Rating Count	0
Price	0
Description	0
Developer	0
Age Rating	0
Languages	60
Size	1
Primary Genre	0
Genres	0
Original Release Date	0
Current Version Release Date	0

dtype: int64

Figure 2.7: Price

```
[ ] data1['Languages'].mode()

0    EN
dtype: object
```

```
[ ] data1['Languages'].fillna(data1['Languages'].mode()[0],inplace=True)
data1.isna().sum()
```

URL	0
ID	0
Name	0
Icon URL	0
Average User Rating	0
User Rating Count	0
Price	0
Description	0
Developer	0
Age Rating	0
Languages	0
Size	1
Primary Genre	0
Genres	0
Original Release Date	0
Current Version Release Date	0

dtype: int64

Figure 2.8: Languages

```
[ ] data1['Size'].median()

56768954.0

[ ] data1['Size'].fillna(data1['Size'].median(),inplace=True)
data1.isna().sum()

URL                                0
ID                                0
Name                              0
Icon URL                          0
Average User Rating               0
User Rating Count                0
Price                            0
Description                       0
Developer                        0
Age Rating                       0
Languages                        0
Size                             0
Primary Genre                    0
Genres                           0
Original Release Date            0
Current Version Release Date     0
dtype: int64
```

Figure 2.9: Size

As we can see from the above figure, our dataset is free from missing values which concludes the preprocessing part.

2.3 Data Exploration:

Data exploration refers to the initial step in data analysis in which data analysts use data visualization and statistical techniques to describe dataset characterizations, such as size, quantity, and accuracy, in order to better understand the nature of the data. Data exploration can use a combination of manual methods and automated tools such as data visualizations, charts, and initial reports. We have used our full dataset for data exploration purpose.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17007 entries, 0 to 17006
Data columns (total 16 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   URL                                    17007 non-null  object
1   ID                                     17007 non-null  int64
2   Name                                  17007 non-null  object
3   Icon URL                             17007 non-null  object
4   Average User Rating                  7560 non-null   float64
5   User Rating Count                    7561 non-null   float64
6   Price                                16983 non-null  float64
7   Description                           17007 non-null  object
8   Developer                            17007 non-null  object
9   Age Rating                           17007 non-null  object
10  Languages                             16947 non-null  object
11  Size                                  17006 non-null  float64
12  Primary Genre                         17007 non-null  object
13  Genres                                17007 non-null  object
14  Original Release Date                 17007 non-null  datetime64[ns]
15  Current Version Release Date          17007 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(4), int64(1), object(9)
memory usage: 2.1+ MB
```

Figure 2.10: Information about dataset

```
plt.figure(figsize=(12,4))
sns.countplot(x='Average User Rating', data=df).set_title("Average User Rating");
```

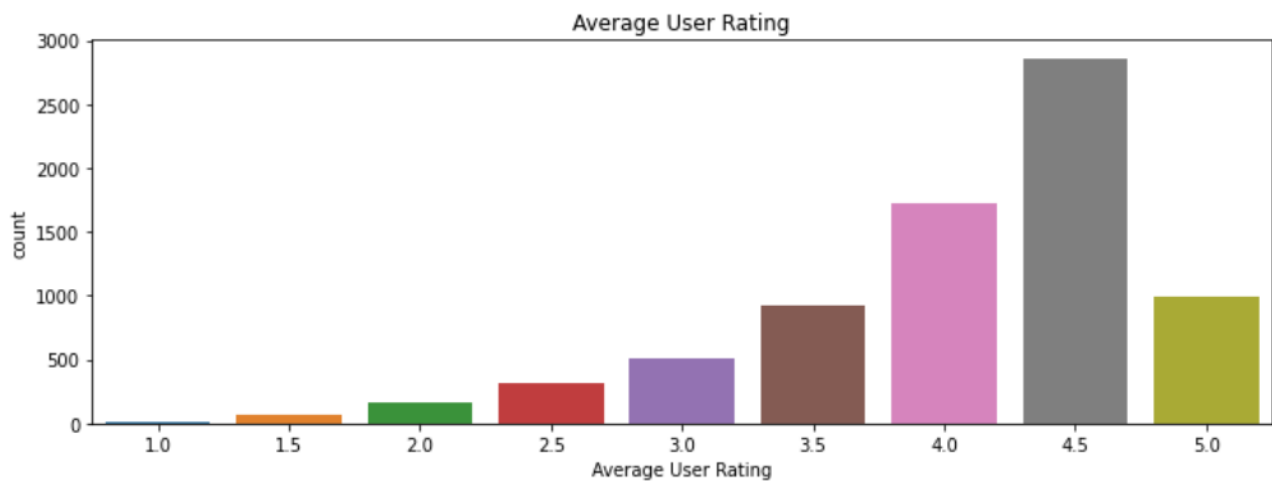


Figure 2.11: Countplot graph 1

In the first graph that is a countplot graph of Average user rating, from the graph we can see that maximum user gives an average of 4.5 ratings to a particular game.

```
plt.figure(figsize=(12,4))
sns.countplot(x='Average User Rating', hue='Age Rating', data=df).set_title("Average User Rating x Age Rating")
```

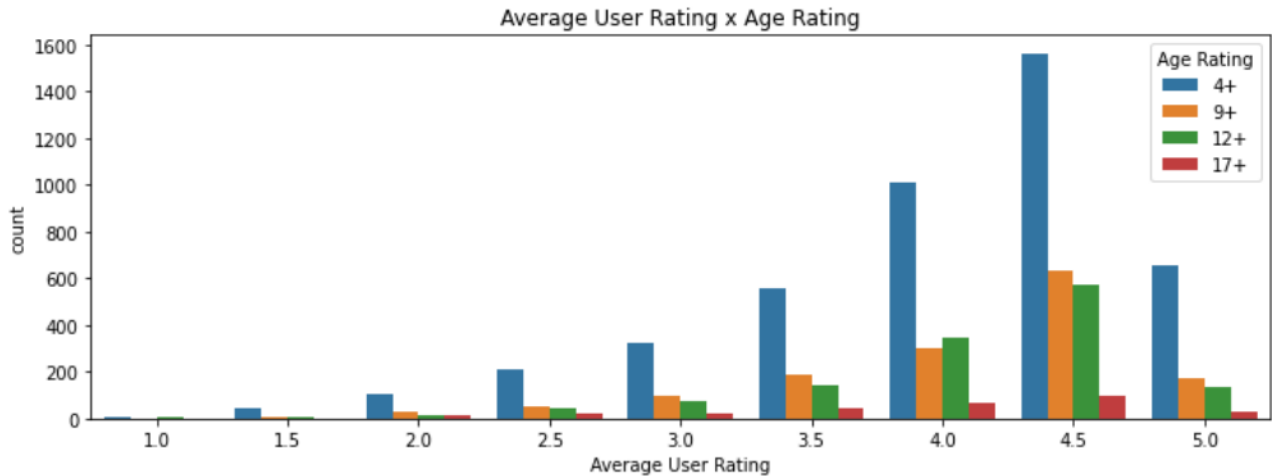


Figure 2.12: Countplot graph 2

Then The Age Rating target is composed of 4 classes, so we are in a context of Multi-Label Classification. We can also analyse the representation of each label in this dataset, in which we can see the maximum number we have is of labelled class 4+.

```
plt.figure(figsize=(6,8))
sns.countplot(y='Primary Genre', data=df).set_title("Primary Genre");
```

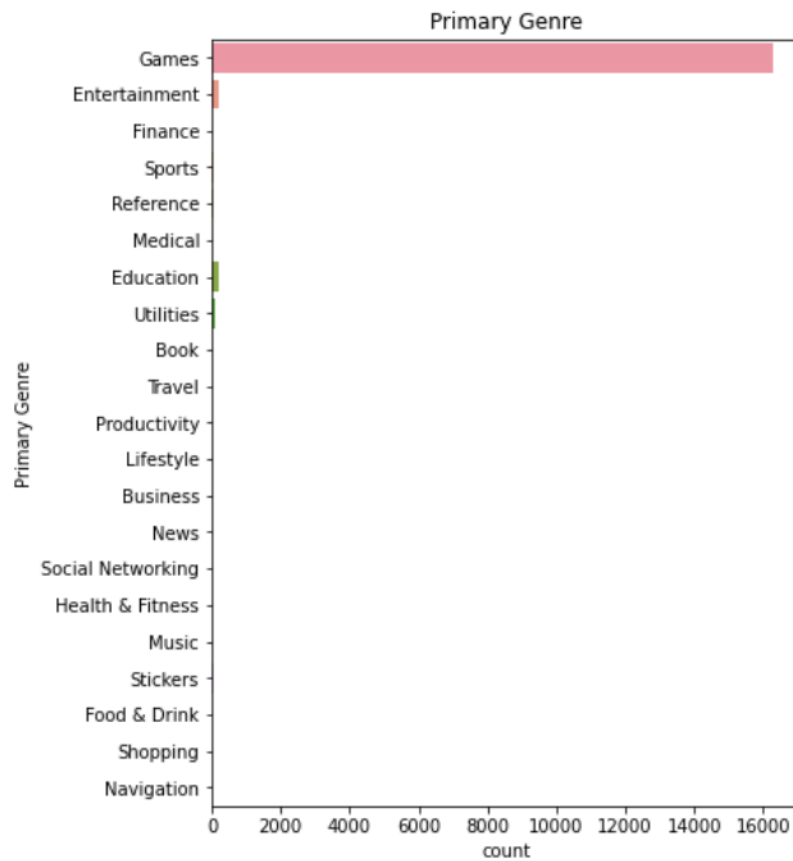


Figure 2.13: Countplot graph 3

This countplot graph shows that the primary genre of maximum dataset values is related to games.


```
Languages = pd.DataFrame(df['Languages'].str.split(',', expand=True).values.ravel(), columns=['Languages'])
sns.countplot(x='Languages', data=Languages, order=pd.value_counts(Languages['Languages']).iloc[:10].index)
```

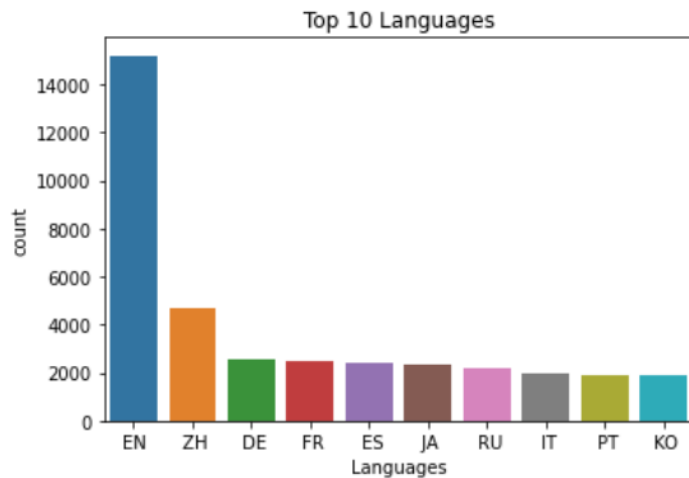


Figure 2.14: Countplot graph 4

This is the language countplot graph ..where we can see most of the users prefer English language more than any other languages mentioned in our dataset.

```
df.Genres.value_counts()

Games, Strategy, Puzzle          778
Games, Puzzle, Strategy          694
Games, Strategy                  588
Games, Strategy, Action          483
Games, Simulation, Strategy      465
...
Games, Strategy, Action, Lifestyle    1
News, Strategy, Action, Games         1
Games, Racing, Education, Strategy    1
Games, Stickers, Board, Emoji & Expressions, Gaming, Strategy  1
Health & Fitness, Strategy, Games, Family  1
Name: Genres, Length: 1004, dtype: int64
```

Figure 2.15: Genre Attribute Information

This is the detailed look in the Genre column.

```
plt.figure(figsize=(6,8))
Genres = pd.DataFrame(df.Genres.str.split(',',expand=True).values.ravel(), columns=['Genre'])
sns.countplot(y='Genre', data=Genres, order=pd.value_counts(Genres['Genre']).iloc[:20].index).set_title("Top 20 Genre")
```

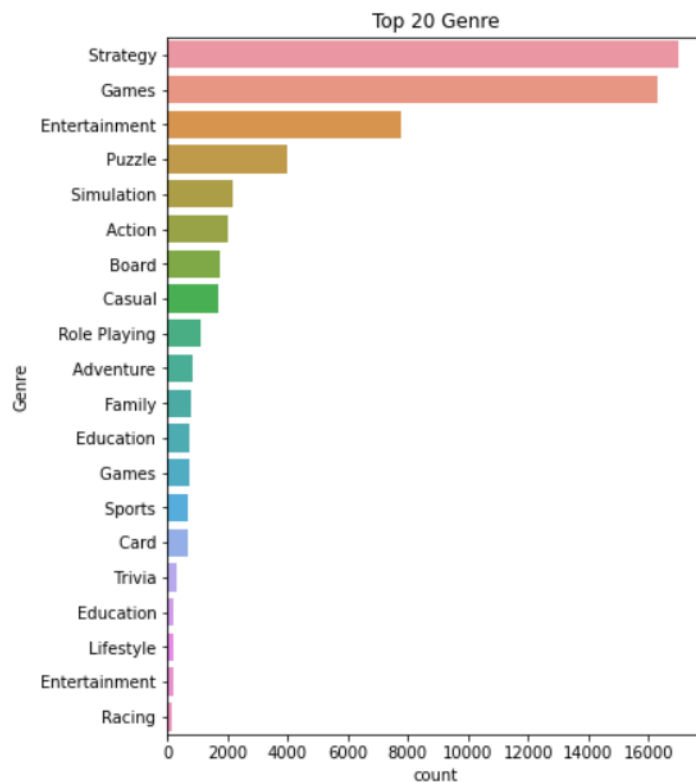


Figure 2.16: Countplot graph 5

In this countplot graph, according to genre attribute information provided we can see an approximate number of 16000 games are related to strategy category.

```

def fast_df_plot(x):
    sns.set()
    df1 = pd.DataFrame(x.dt.year.value_counts()).reset_index()
    return sns.lineplot(x=df1.iloc[:,0], y=df1.iloc[:,1]).set_title("Original Release Date x Year")

fast_df_plot(df['Original Release Date']);

```

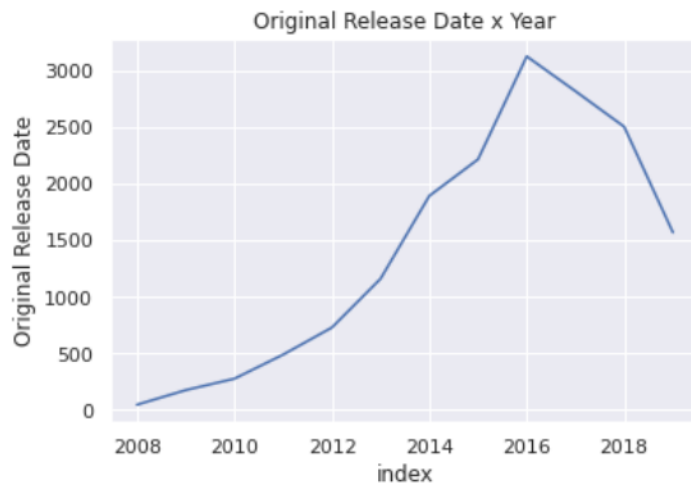


Figure 2.17: Lineplot graph 1

This is a lineplot graph, here we can see the release dates of a game plotted against the year and thus we can see that in the year 2016 majority games were released.

```
def fast_df_plot(x):
    sns.set()
    df1 = pd.DataFrame(x.dt.year.value_counts()).reset_index()
    return sns.lineplot(x=df1.iloc[:,0], y=df1.iloc[:,1]).set_title("Current Version Release Date")

fast_df_plot(df['Current Version Release Date']);
```

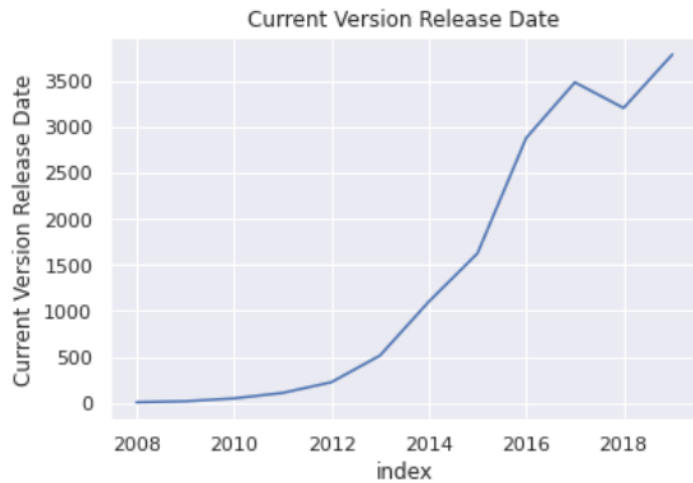


Figure 2.18: Lineplot graph 2

In this Lineplot graph we can see the dates wen the recent update to the graph was released.

```
def fast_df_plot(x):
    sns.set()
    df1 = pd.DataFrame(x.dt.year.value_counts()).reset_index()
    return sns.lineplot(x=df1.iloc[:,0], y=df1.iloc[:,1]).set_title("Current Version Release Date")

fast_df_plot(df['Current Version Release Date']);
```

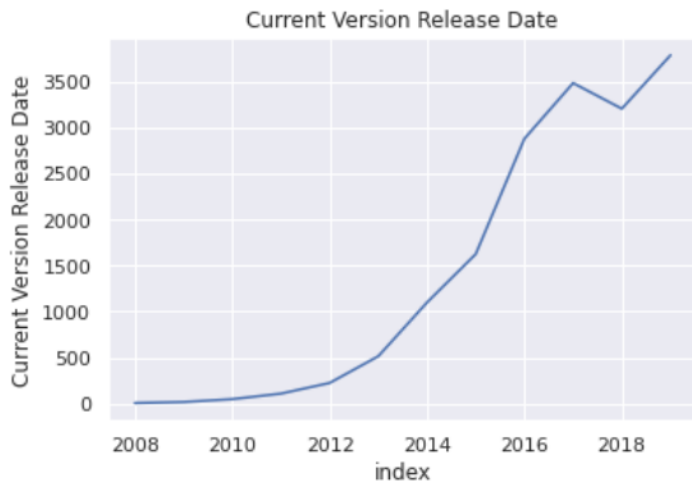


Figure 2.19: Lineplot graph 3

Here we have plotted a lineplot diagram for size against original release date.

```
plt.figure(figsize=(12,8))
sns.lineplot(x='Original Release Date', y='Size', data=df).set_title("Original Release Date x Size");
```

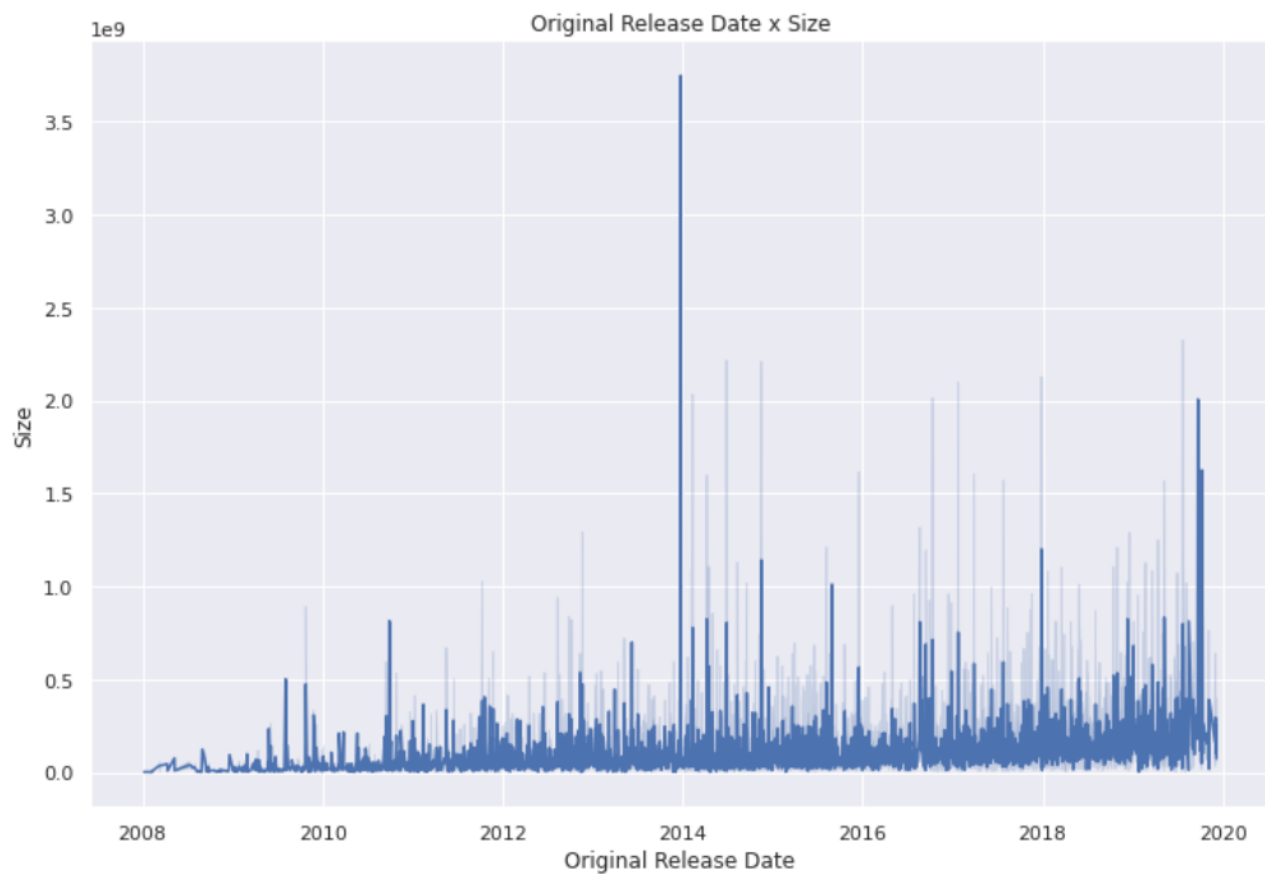


Figure 2.20: Lineplot graph 4

```
plt.figure(figsize=(12,8))
sns.lineplot(x='Current Version Release Date', y='Size', data=df).set_title("Current Version Release Date
```

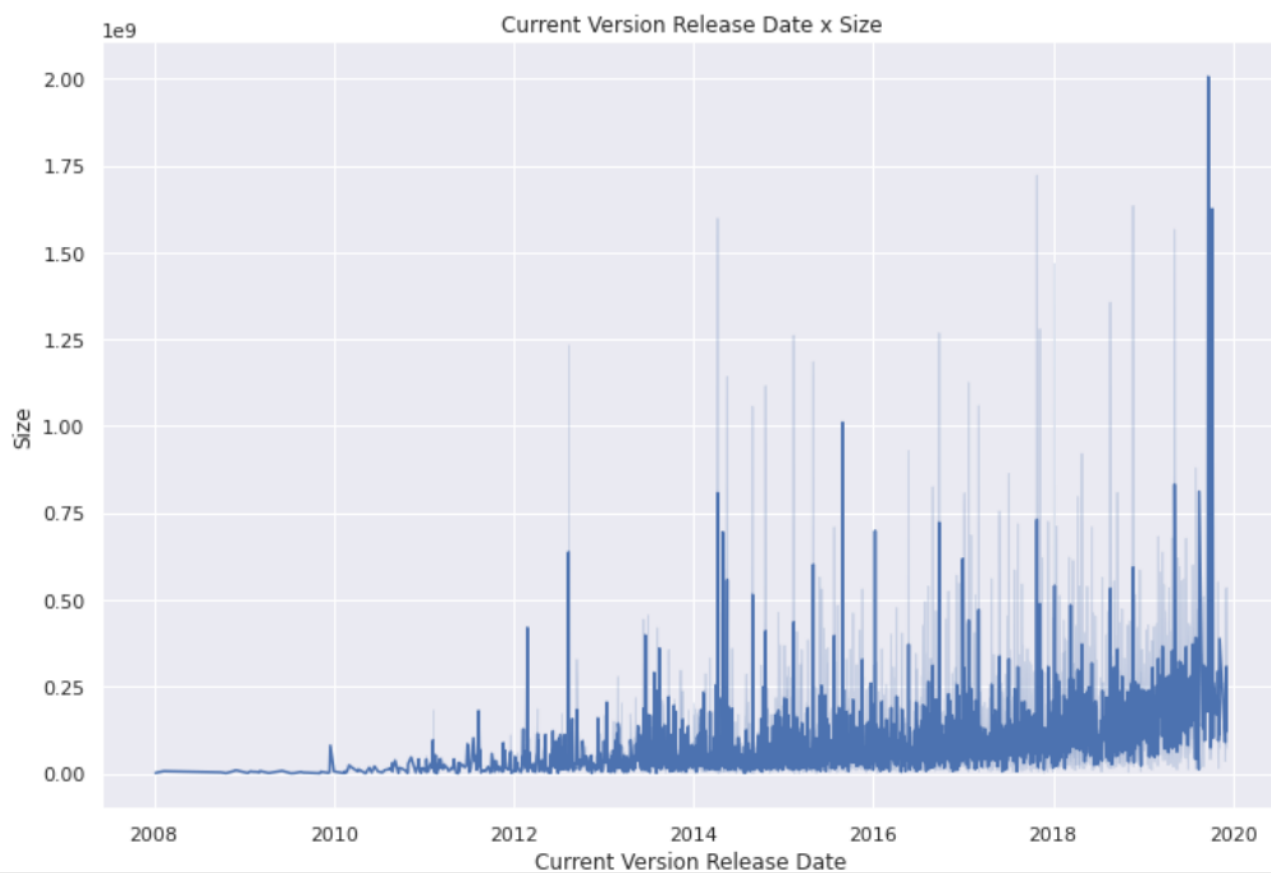


Figure 2.21: Lineplot graph 5

```
sns.scatterplot(x='Size', y='Average User Rating', data=df).set_title('Size x Average User Rating');
```

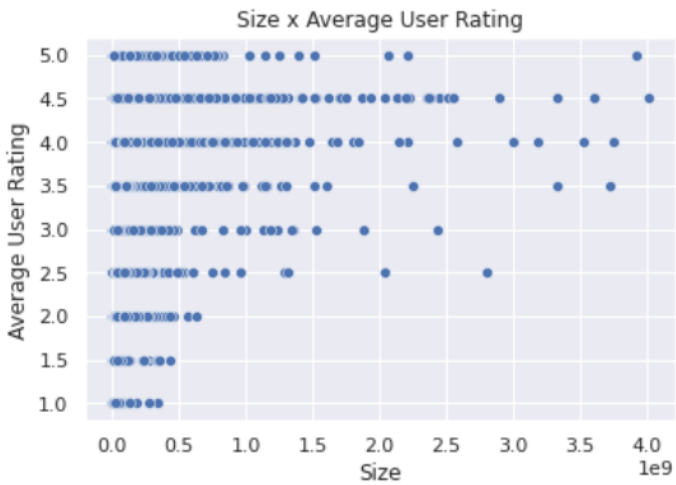


Figure 2.22: Scatterplot graph 1

```
sns.scatterplot(x='Size', y='Price', data=df).set_title('Size x Price');
```



Figure 2.23: Scatterplot graph 2


```
sns.scatterplot(x='Price', y='Average User Rating', data=df).set_title('Price x Average User Rating');
```

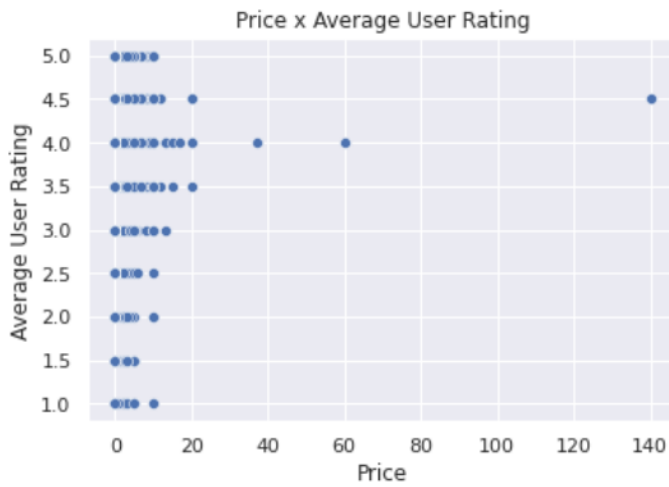


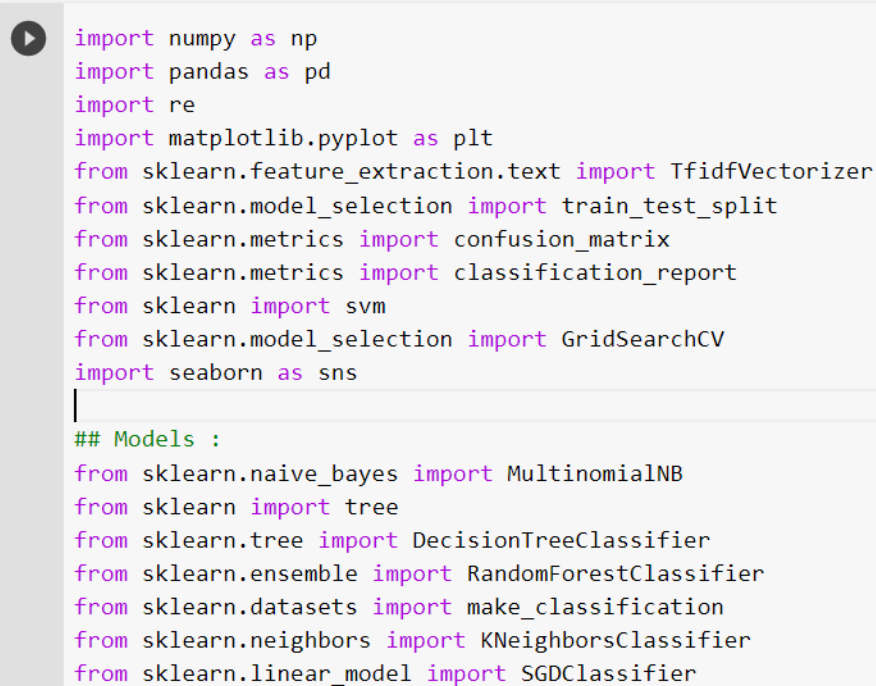
Figure 2.24: Scatterplot graph 3

Lastly we have plotted scatterplots for Size against Average User rating and Price and Price against Average User Rating.

2.4 Classification Algorithms:

Classification means arranging the mass of data into different classes or groups on the basis of their similarities and resemblances. Classification plays an integral role in the context of mining techniques. As suggested by its name, this is a process where you classify data. And, many decisions need to be made to bring the data together. Often, it depends on a set of input variables. The classification depends on a series of acknowledgements and data instances.

1. Importing libraries



```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import seaborn as sns

## Models :
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
```

Figure 2.25: Importing libraries

```
[ ] pd.set_option('display.max_columns', None) # allows to display all columns of the df
```

```
[ ] import warnings # ignore warnings
    warnings.filterwarnings('ignore')
```

```
[ ] df=data1.copy()
    df[:3]
```

	URL	ID	Name	Icon URL	Average User Rating
0	https://apps.apple.com/us/app/sudoku/id284921427	284921427	Sudoku	https://is2-ssl.mzstatic.com/image/thumb/Purpl...	4.0
1	https://apps.apple.com/us/app/reversi/id284926400	284926400	Reversi	https://is4-ssl.mzstatic.com/image/thumb/Purpl...	3.5
2	https://apps.apple.com/us/app/morocco/id284946595	284946595	Morocco	https://is5-ssl.mzstatic.com/image/thumb/Purpl...	3.0

Figure 2.26: Displaying data

As we only need our two variables of interest, we can drop the others.

```
df = df.filter(["Age Rating", "Description"])
df[:3]
```

	Age Rating	Description
0	4+	Join over 21,000,000 of our fans and download ...
1	4+	The classic game of Reversi, also known as Oth...
2	4+	Play the classic strategy game Othello (also k...

Figure 2.27: Dropping columns

Before beginning the pre-processing phase, we can just quickly have a look at the observations, with some descriptive statistics.

```

▶ df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17007 entries, 0 to 17006
Data columns (total 2 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Age Rating      17007 non-null  object
1   Description     17007 non-null  object
dtypes: object(2)
memory usage: 265.9+ KB

```

Figure 2.28: Displaying columns

2. Pre-Processing Phase(Cleaning of the Description feature): As we will be able to see by printing some descriptions, there is some cleaning to do : we want to remove the reccurent patterns (for example "n n") that bring us absolutely no more informations to predict the Age Rating. We can clean this up thanks to the use of Regular Expressions (RE package for python). After some exploring, here is a list of the elements to remove :

- (a) reccurent patterns probably due to scrapping : slash n , u
- (b) categories of the description in the App Store : OPTIONS, FEATURES
- (c) symbols and punctuation : *, ., ; ,],) , etc
- (d) web site links : "www.[a-z]*.[a-z]2"
- (e) years : "[2][0-9]3"

```

[ ] df.Description[0]

'Join over 21,000,000 of our fans and download one of our Sudoku games today!\n\nMakers of the Best Sudoku Game of 2008, Sudoku
iPhone with great features and 1000 unique puzzles! \n\nSudoku will give you many hours of fun and puzzle solving. Enjoy the c
you are using your iPhone or iPod Touch. \n\nOPTIONS\n\nAll options are on by default, but you can turn them off in the Opti
swers in red. \n\nSmart Buttons :: Disables the number button when that number is completed on the game board. \n\n
the box, column, and row that contains the cell with your correct answer.\n\nFEATURES\n\n1000 unique handcrafted puz
\n\nFour different skill levels\n\nChallenge a friend\n\nMultiple...'

```

Figure 2.29: Displaying Description

```

# Remove some recurrent patterns with regular expressions :

list_expr_to_remove = [r"\\n",r"OPTIONS",r"FEATURES",
                        r"\\u",r"[2][0-9]{3}",r"\\",
                        r"www.[a-z]*.[a-z]{2}"]

for expr in list_expr_to_remove :
    df.Description = df.Description.apply(lambda x : re.sub(expr,"",x))

# Remove all symbols and punctuation :

list_punctuation_symbols = [".", ",", "!", "?", "(", ")", "[", "]", "*", ":", "'", "-"]
|
for elem in list_punctuation_symbols :
    df.Description = df.Description.apply(lambda x : x.replace(elem,""))

# For each description, change all letters from upper case to lower case :
df.Description = df.Description.apply(lambda x : x.lower())

[ ] df.Description[0]

'join over 21000000 of our fans and download one of our sudoku games todaymakers of the best sudoku game of  sudoku free we offer
t features and 1000 unique puzzles sudoku will give you many hours of fun and puzzle solving enjoy the challenge of solving sudoko
ne or ipod touch all options are on by default but you can turn them off in the options menu show incorrect  shows incorrect ans
when that number is completed on the game board  smart notes  removes the number from the notes in the box column and row that
handcrafted puzzles all puzzles solvable without guessing four different skill levels challenge a friend multiple color schemes
ossible answers for each square  tap the all notes button off to remove the notes hi...'

```

Figure 2.30: Displaying Description after cleaning

Now we have to turn text data into numerical data, that will be the aim of the next section.

3. TF-IDF Method : turn descriptions into vectors. The TF-IDF method is a really simple alternative in a context of classification when using texts as features. Formally, we want to turn text data into numerical data so Machine Learning models can clearly interpret the feature in input. To proceed, we need to identify in each text what are the most relevant words, i.e that bring us the most information to put a text in a certain category.

```

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df.Description)

[ ] X[0]

<1x129803 sparse matrix of type '<class 'numpy.float64''>'
with 128 stored elements in Compressed Sparse Row format>

```

Figure 2.31: Vectorizer method

4. Train And split data Next is the modelisation phase. First we need to split our data into two subsets :

- the train set to fit the models
- the test set to evaluate the models

```
[ ] X.shape
```

```
(17007, 129803)
```

```
[ ] y = df["Age Rating"].to_numpy()  
y = y.reshape(y.shape[0],1)  
y.shape
```

```
(17007, 1)
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

Figure 2.32: Train And split data

2.4.1 Naive Bayes Classifier:

Naive Bayes Classifier is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. So we here assume that all descriptions of the dataset are independent (it is quite credible for our situation). We will not go into details but you just have to know that this classifier will be able to choose a category for a description by maximizing the conditionnal probability of being in a class, given the informations of the description (the words observed in the description).

Naïve Bayes classifier is basically a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, and prediction is done wrt to a target variable which here is age rating.

```
[ ] naive_bayes = MultinomialNB()  
naive_bayes.fit(X_train, y_train)  
  
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Figure 2.33: Naïve Bayes classifier model

```

y_pred = naive_bayes.predict(X_test)
target_names = np.unique(list(df["Age Rating"].values))

print("Classification Report : ")
print(classification_report(y_test, y_pred, target_names = target_names))
print("-----")
print("Confusion Matrix : ")
print(confusion_matrix(y_test, y_pred))

```

```

Classification Report :
              precision    recall  f1-score   support

    12+         0.36         0.01         0.02         402
    17+         0.00         0.00         0.00         123
     4+         0.70         1.00         0.82        2364
     9+         0.00         0.00         0.00         513

 accuracy                   0.69         3402
 macro avg              0.26         0.25         0.21         3402
 weighted avg           0.53         0.69         0.57         3402

-----
Confusion Matrix :
[[  4   0 397   1]
 [  1   0 121   1]
 [  5   0 2355   4]
 [  1   0 512   0]]

```

Figure 2.34: Naïve Bayes output

As we can see thanks to the classification report, the accuracy of this model is 69%. It is quite a good score, since we notice that we didn't specify the hyperparameters for this model. Moreover, if we observe the confusion matrix, we see that there remain many "4+" observations that were predicted as "12+" (397), "17+" (121) or "9+" (512). Perhaps it is possible to improve the quality of this model, by tuning the values of the hyperparameters. To proceed, we will use GridSearchCV, that allows us to select a list of hyperparameters, and then will test and select the best combination of parameters (in terms of accuracy).

```

▶ parameters = {'alpha' : np.arange(0,5,0.1),
                'fit_prior' : (True, False)}

naive_bayes = MultinomialNB()
clf_naive_bayes = GridSearchCV(naive_bayes, parameters)
clf_naive_bayes.fit(X_train, y_train)

☞ GridSearchCV(cv=None, error_score=nan,
               estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                       fit_prior=True),
               iid='deprecated', n_jobs=None,
               param_grid={'alpha': array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])},
               'fit_prior': (True, False)},
               pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
               scoring=None, verbose=0)

```

Figure 2.35: Tuning hyperparameters

```

[ ] clf_naive_bayes.best_score_

0.7476662991547225

```

Figure 2.36: Best Score (Naïve Bayes)

Hence, with the help of tuning we improved the accuracy of the model by 5%.

2.4.2 Decision Tree Classifier:

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. Since we need to do Multi-label Classification, we will use the DecisionTreeClassifier model of sklearn.

```

▶ decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

☞ DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=None, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort='deprecated',
                        random_state=None, splitter='best')

```

Figure 2.37: Decision Tree Classifier model


```

▶ y_pred = decision_tree.predict(X_test)
  target_names = np.unique(list(df["Age Rating"].values))

  print("Classification Report : ")
  print(classification_report(y_test, y_pred, target_names = target_names))

  print("-----")
  print("Confusion Matrix : ")
  print(confusion_matrix(y_test, y_pred))

```

```

↳ Classification Report :
              precision    recall  f1-score   support

    12+         0.36         0.37         0.37         402
    17+         0.30         0.25         0.28         123
     4+         0.84         0.86         0.85        2364
     9+         0.38         0.35         0.36         513

 accuracy          0.70         0.70         0.70        3402
 macro avg         0.47         0.46         0.46        3402
 weighted avg      0.69         0.70         0.70        3402

-----
Confusion Matrix :
[[ 150   15  145   92]
 [   21   31   49   22]
 [  128   34 2033  169]
 [   118   22  196  177]]

```

Figure 2.38: Decision Tree Output

```

[ ] decision_tree.score(X_test, y_test)

0.7028218694885362

```

Figure 2.39: Best Score (Decision Tree Classifier)

The score here is nearly the same that for the previous model Naive Bayes (without tuning). However, the confusion matrix is very different : precision is better for the less represented classes than the Naive Bayes model.

2.4.3 K-Nearest Neighbors Classifier:

KNN is an approach to data classification that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in.

This KNN model is perhaps the most adapted for our situation : indeed, the aim of this process is to determine the K-closest observations of each input (given K the number of observations to determine, and some metrics that define the distance evaluation to find the "neighbors"). Then the model will simply decide to classify the input in the class that most appears amongs its "neighbors" class.

```
[ ] knn = KNeighborsClassifier()
    knn.fit(X_train, y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Figure 2.40: K-Nearest Neighbors Classifier model

```
[ ] y_pred = knn.predict(X_test)
    target_names = np.unique(list(df["Age Rating"].values))

    print("Classification Report : ")
    print(classification_report(y_test, y_pred, target_names = target_names))

    print("-----")
    print("Confusion Matrix : ")
    print(confusion_matrix(y_test, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

    12+         0.40         0.44         0.42         402
    17+         0.53         0.36         0.43         123
     4+         0.83         0.92         0.88        2364
     9+         0.52         0.28         0.36         513

 accuracy                   0.75         3402
 macro avg         0.57         0.50         0.52         3402
 weighted avg         0.72         0.75         0.73         3402

-----
Confusion Matrix :
[[ 175   14  147   66]
 [  25   44   48    6]
 [ 111   18 2175   60]
 [ 129    7  235  142]]
```

Figure 2.41: K-Nearest Neighbors Classifier Output

```
[ ] knn.score(X_test, y_test)

0.7454438565549677
```

Figure 2.42: Best Score (K-Nearest Neighbors Classifier)

The score of this kind of model is better than the two others (without tuning the parameters). Instead of tuning with SearchGridCV, we can just have a graphic representation of the impact of the number of neighbors on the score of the model.

```
[ ] scores_list = []
n_neighbors_list = [2,5,10,20,30,40,50,60,70,80,90,100,120,140,180,200]
for n_neighbors_value in n_neighbors_list:
    knn = KNeighborsClassifier(n_neighbors = n_neighbors_value)
    knn.fit(X_train, y_train)
    scores_list.append(knn.score(X_test, y_test))
```



```
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)

sns.set_theme(style = "whitegrid")
ax = sns.lineplot(x = n_neighbors_list, y = scores_list, color = "green")
ax.set(xlabel = "n_neighbors", ylabel = "KNN Model Score")
plt.xticks(rotation = 0, size = 12, weight = 'normal')
plt.title("Evolution of the KNN Model Score depending of the number of neighbors hyperparameter
          \n", fontsize=18, color='#009432')
plt.show()
```

Figure 2.43: Tuning of KNN Model

[] Evolution of the KNN Model Score depending of the number of neighbors hyperparameter

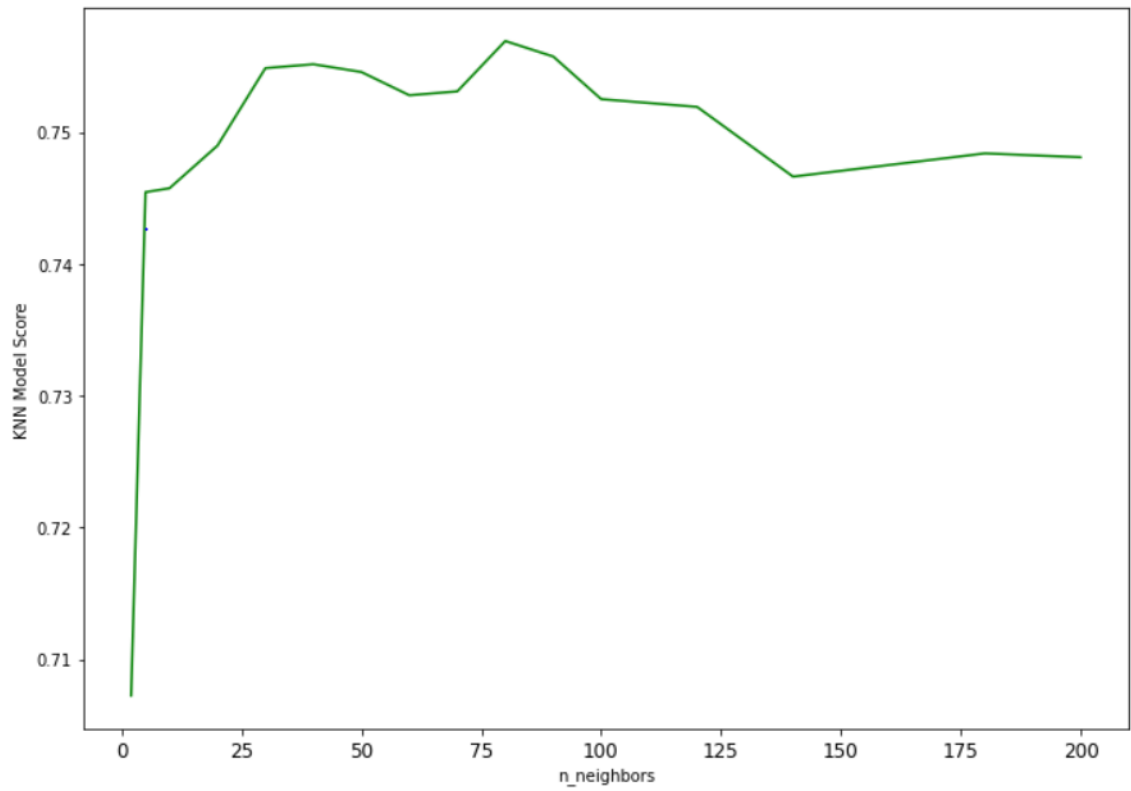


Figure 2.44: Graph of impact of the number of neighbors on the score of the model

By a simple analysis of the graphic, we can see that the optimal number of neighbors is close to 80 (without touching the others parameters).

```
[ ] knn = KNeighborsClassifier(n_neighbors = 80)
    knn.fit(X_train, y_train)
    knn.score(X_test, y_test)
```

0.7569077013521458

Figure 2.45: Improved Accuracy of KNN

Hence, by tuning we have successfully increased the accuracy.

2.5 Clustering Algorithms:

Clustering is the process of making a group of abstract objects into classes of similar objects.

2.5.1 Kmeans Clustering:

K-Means clustering intends to partition n objects into k clusters in which each object belongs to the cluster with the nearest mean. This method produces exactly k different clusters of greatest

possible distinction. The best number of clusters k leading to the greatest separation (distance) is not known a priori and must be computed from the data. The objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function.

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
data2 = data1[["ID", "Price"]]
data2.head()
data2 = data2.head(15)
print(data2.head())
plt.scatter(data2['ID'], data2['Price'])
plt.show()
scaler = MinMaxScaler()
scaler.fit(data2[['Price']])
data2['Price'] = scaler.transform(data2[['Price']])
scaler.fit(data2[['ID']])
data2['ID'] = scaler.transform(data2[['ID']])
print(data2)
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(data2[['ID', 'Price']])
data2['cluster'] = y_predicted
print(data2)
df1 = data2[data2.cluster==0]
df2 = data2[data2.cluster==1]
df3 = data2[data2.cluster==2]
plt.scatter(df1.ID, df1['Price'], color="green")
plt.scatter(df2.ID, df2['Price'], color="red")
plt.scatter(df3.ID, df3['Price'], color="black")
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='purple', marker='*', label='centroID')
plt.xlabel('ID')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 2.46: Code for Kmeans Clustering

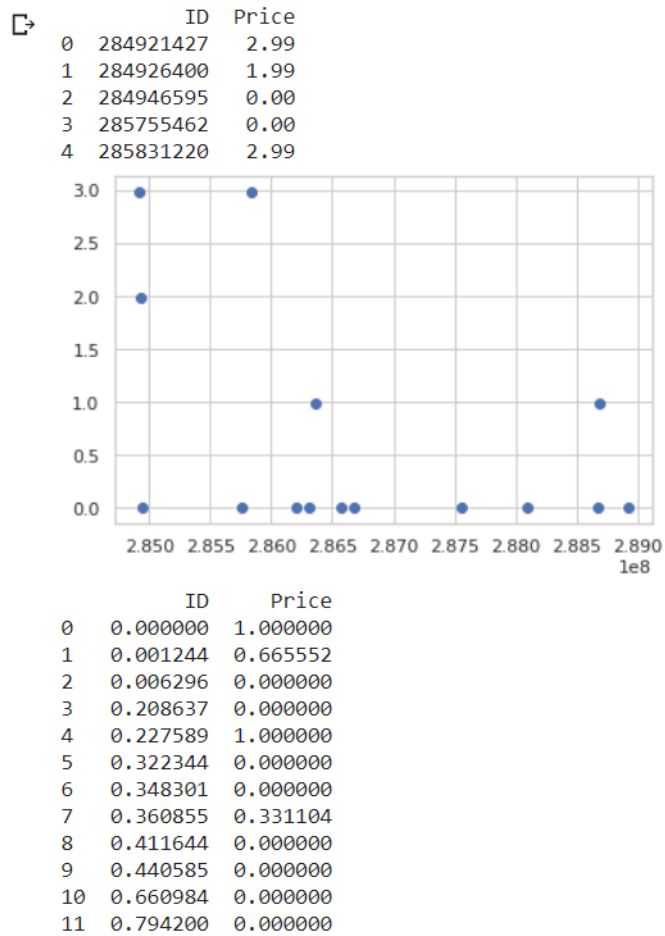


Figure 2.47: Before applying kmeans

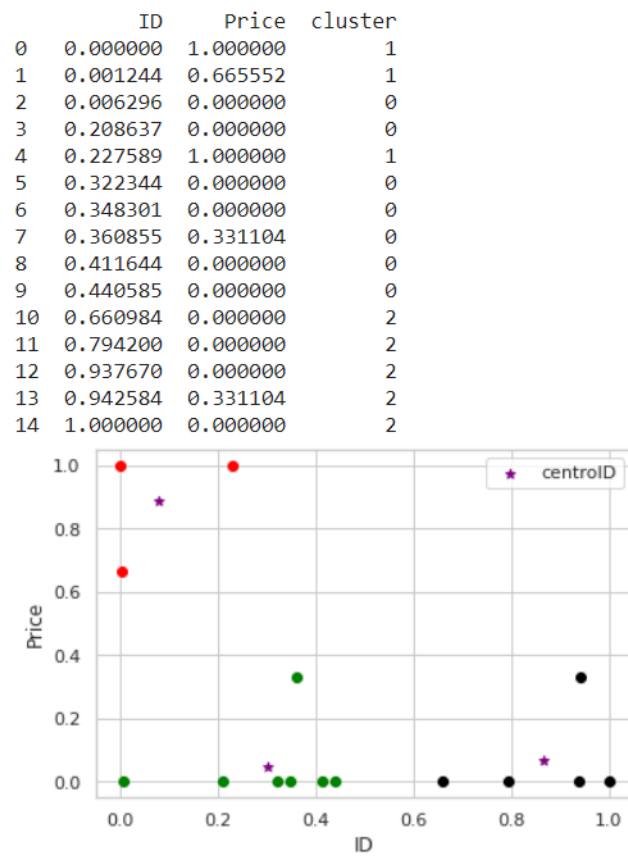


Figure 2.48: After applying kmeans

2.5.2 DBSCAN Clustering:

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

```
[ ] import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN
from collections import Counter
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
from pylab import rcParams
rcParams['figure.figsize']=14, 6

%matplotlib inline
```

Figure 2.49: Importing libraries for DBSCAN

```
[ ] data3=data1.copy()
_= plt.plot(data3['Average User Rating'], data3['Price'],
            marker='.', linewidth=0, color='#128128')
_= plt.grid(which='major', color='cccccc', alpha=0.45)
_= plt.title('Graph plot', family='Arial', fontsize=12)
_= plt.xlabel('Average User Rating')
_= plt.ylabel('Price')
_= plt.show()
```

findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.

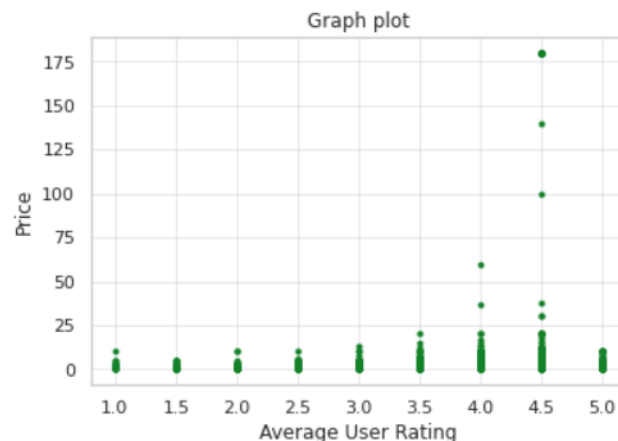


Figure 2.50: Plotting graph before applying clustering


```
[ ] #DBSCAN MODEL
db_data=data3[['Average User Rating','Price']]
db_data=db_data.values.astype('float32',copy=False)
db_data #numpy array

array([[4.  , 2.99],
       [3.5 , 1.99],
       [3.  , 0.  ],
       ...,
       [4.5 , 0.  ],
       [4.5 , 0.  ],
       [4.5 , 0.  ]], dtype=float32)

[ ] db_data_scaler=StandardScaler().fit(db_data)
db_data=db_data_scaler.transform(db_data)
db_data #normalized numpy array

array([[ -0.5578996 ,  0.27812526],
       [-1.4729539 ,  0.15041183],
       [-2.3880084 , -0.10373789],
       ...,
       [ 0.35715473, -0.10373789],
       [ 0.35715473, -0.10373789],
       [ 0.35715473, -0.10373789]], dtype=float32)

[ ] #Construct Model
model = DBSCAN(eps = 0.25, min_samples = 12, metric='euclidean').\
fit(db_data)
model #Prepared DBSCAN model

DBSCAN(algorithm='auto', eps=0.25, leaf_size=30, metric='euclidean',
        metric_params=None, min_samples=12, n_jobs=None, p=None)
```

Figure 2.51: Preparing Model

```
[.] outliers_df = data3[model.labels_ == -1] #-1 stands for OUTLIERS
clusters_df = data3[model.labels_ != -1]

colors = model.labels_
colors_clusters = colors[colors != -1]
color_outliers = 'black'

clusters = Counter(model.labels_)
print(clusters)
print(data[model.labels_ == -1].head())
print('Number of clusters = {}'.format(len(clusters)-1))
```

```
Counter({3: 12205, 0: 1714, 6: 980, 1: 915, 2: 510, 4: 314, 5: 154, -1: 74, 8: 58, 7: 53, 9: 30})
```

	URL	ID \
66	https://apps.apple.com/us/app/omar-sharif-brid...	304878580
185	https://apps.apple.com/us/app/king-of-dragon-p...	335545504
276	https://apps.apple.com/us/app/smartgo-kifu/id3...	364854741
364	https://apps.apple.com/us/app/imperial/id38267...	382679047
513	https://apps.apple.com/us/app/romance-of-the-t...	408829853

	Name \
66	Omar Sharif Bridge
185	King of Dragon Pass
276	SmartGo Kifu
364	Imperial
513	ROMANCE OF THE THREE KINGDOMS\u3000\u3000

	Icon URL	Average User Rating \
66	https://is2-ssl.mzstatic.com/image/thumb/Purpl...	2.5
185	https://is1-ssl.mzstatic.com/image/thumb/Purpl...	5.0
276	https://is4-ssl.mzstatic.com/image/thumb/Purpl...	4.5
364	https://is2-ssl.mzstatic.com/image/thumb/Purpl...	3.5
513	https://is3-ssl.mzstatic.com/image/thumb/Purpl...	3.5

Figure 2.52: Visualizing results-1

	Description \
66	Welcome to the 2019 edition of Omar Sharif Bri...
185	"Create your own epic saga of conflict, mythol...
276	"SmartGo Kifu transforms your iPad into a Go b...
364	iPad version of the brilliant Imperial board g...
513	A new version of the app has been released.\nT...

	Developer	Age Rating \
66	ZingMagic Limited	4+
185	A Sharp, LLC	12+
276	Smart Go, Inc.	4+
364	Javelin Sdn Bhd	9+
513	KOEI TECMO GAMES CO., LTD.	4+

	Languages	Size	Primary Genre \
66	EN	33982464.0	Games
185	EN	364954624.0	Games
276	EN, FR, DE, JA, KO, RU, ZH, ES, ZH	64207872.0	Games
364	EN	17991680.0	Games
513	EN, JA, ZH, ZH	423871488.0	Games

	Genres	Original Release Date \
66	Games, Card, Entertainment, Strategy	17-02-2009
185	Games, Entertainment, Strategy, Role Playing	14-02-2011
276	Games, Strategy, Board, Entertainment	01-04-2010
364	Games, Strategy, Board	30-09-2011
513	Games, Entertainment, Simulation, Strategy	22-12-2010

	Current Version	Release Date
66		11-12-2018
185		18-10-2018
276		17-04-2019
364		23-03-2018
513		07-01-2016

Number of clusters =10

Figure 2.53: Visualizing results-2

```
[ ] fig= plt.figure()
    ax=fig.add_axes([.1,.1, 1, 1])
    ax.scatter(clusters_df['Average User Rating'], clusters_df['Price'],
               c=colors_clusters, edgecolors='black',s=50)
    ax.scatter(outliers_df['Average User Rating'], outliers_df['Price'],
               c=color_outliers, edgecolors='black',s=50)

    ax.set_xlabel('Average User Rating', family='Arial', fontsize=10)
    ax.set_ylabel('Price', family='Arial', fontsize=10)
    plt.title('DBSCAN model', family='Arial', fontsize=12)
    plt.grid(which='major', color='#cccccc', alpha=0.45)
    plt.show()    #10 clusters and data noise(outliers)
```

Figure 2.54: Plotting clusters and outliers

findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.

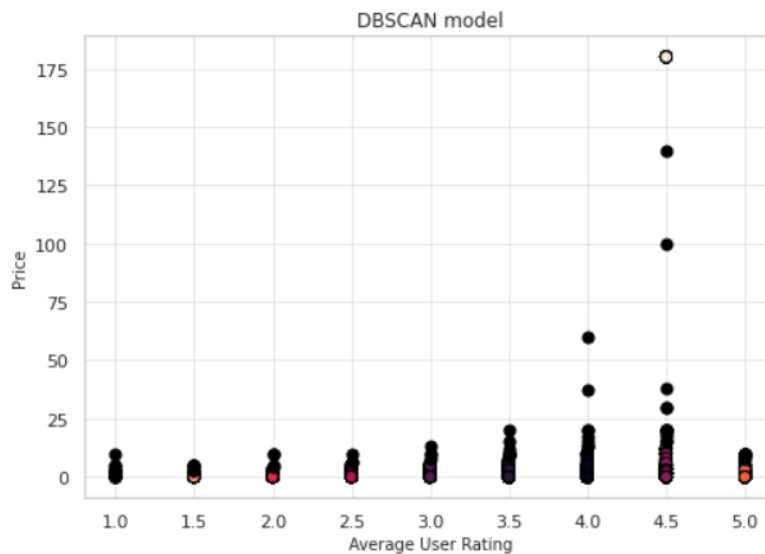


Figure 2.55: DBSCAN Output

Conclusion

We've tested three different machine learning models who were able to predict the age rating of a game app by simply interpreting its description on the app store. The accuracies obtained are quite encouraging and are as follows:

- 74.77 % for the Multinomial Naive Bayes Classifier
- 70.28 % for the Decision Tree Classifier
- 75.69 % for the K-Nearest Neighbors Classifier

Hence we could predict the age rating of the Appstore games with the help of the description provided with the highest accuracy of 75.69%.

References

1. <https://www.kaggle.com/tristan581/17k-apple-app-store-strategy-games>
2. <https://www.youtube.com/watch?v=eq1zKgCFwkk&t=306s>
3. [https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html#:~:text=low%20point%20density.-,Density%2DBased%20Spatial%20Clustering%20of%20Applications%20with%20Noise%20\(DBSCAN\),is%20containing%20noise%20and%20outliers.](https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html#:~:text=low%20point%20density.-,Density%2DBased%20Spatial%20Clustering%20of%20Applications%20with%20Noise%20(DBSCAN),is%20containing%20noise%20and%20outliers.)
4. <https://www.javatpoint.com/data-mining-techniques>
5. <https://www.geeksforgeeks.org/basic-concept-classification-data-mining/#:~:text=In%20the%20process%20of%20data,distinguishes%20data%20classes%20and%20concepts.>